Can Frogs Find Large Independent Sets in a Decentralized Way? Yes They Can!

Christian Blum[1,2], Maria J. Blesa[3], and Borja Calvo[1]

[1] Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, San Sebastian, Spain
{christian.blum,borja.calvo}@ehu.es
[2] IKERBASQUE, Basque Foundation for Science, Bilbao, Spain
[3] ALBCOM Research Group, Universitat Politécnica de Catalunya, Barcelona, Spain
mjblesa@lsi.upc.edu

Abstract. The problem of identifying a maximal independent (node) set in a given graph is a fundamental problem in distributed computing. It has numerous applications, for example, in wireless networks in the context of facility location and backbone formation. In this paper we study the ability of a bio-inspired, distributed algorithm, initially proposed for graph coloring, to generate large independent sets. The inspiration of the considered algorithm stems from the self-synchronization capability of Japanese tree frogs. The experimental results confirm, indeed, that the algorithm has a strong tendency towards the generation of colorings in which the set of nodes assigned to the most-used color is rather large. Experimental results are compared to the ones of recent algorithms from the literature. Concerning solution quality, the results show that the frog-inspired algorithm has advantages especially for the application to rather sparse graphs. Concerning the computation round count, the algorithm has the advantage of converging within a reasonable number of iterations, regardless of the size and density of the considered graph.

1 Introduction

Given an undirected graph $G = (V, E)$, an independent set is a subset of the nodes of G such that no two nodes of this set are connected by an edge e from E. Furthermore, a *maximal independent set* $V_{\text{MIS}} \subseteq V$ is an independent set such that no other independent set $\hat{V} \subseteq V$ exists with $V_{\text{MIS}} \subseteq \hat{V}$. In other words, it is—per definition—not possible to add an additional node to a maximal independent set without destroying its independent set property. Finally, a *maximum independent set* is a maximal independent set of maximal size. Both the maximum independent set problem and the maximal independent set problem are fundamental in computer science and related fields (see, for example, [4]). From the perspective of centralized algorithms, it is well known that the maximum independent set problem is NP-hard [6], while the maximal independent set problem is in P. In fact, the literature offers various, rather simple, greedy algorithms for the generation of maximal independent sets (see, for example, [7]).

In this work we consider the maximal independent set problem in a distributed setting (henceforth labelled MIS). This problem has applications, for example, in the context of facility location and backbone formation in wireless networks [5]. In particular, we focus on the study of a recently proposed (distributed) algorithm for graph coloring, named FrogSim (see [8, 9, 1]). The experimental results show that FrogSim, which is an algorithm inspired by the self-desynchronization of the calls of male Japanese tree frogs, has the tendency to generate color assignments in which the nodes associated to the most-used color correspond to large independent sets. The results obtained by FrogSim are compared to one of the most-recently proposed distributed algorithms for the MIS problem, which was initially published in the prestigious journal *Science* [2]. More specifically, the results are compared to an optimized version of this algorithm from [11, 10].

The reminder of this article is organized as follows. In Section 2, a short description of the studied algorithm is provided. The experimental evaluation is documented in Section 3. Finally, conclusions and an outlook to future work can be found in Section 4.

2 The FrogSim Algorithm

Even though a description of the FrogSim algorithm can be found, for example, in [9], in the following we provide a—rather short—description in order for the paper to be self-contained. The following algorithm description is thought for working in a static wireless ad hoc network with n nodes equipped with radio antennas. Depending on the type of antennas and their communication range, a communication graph is implicitly defined in which each edge indicates a pair of nodes for which node-to-node communication is possible.

2.1 Algorithm Preliminaries

A first, preliminary, step requires an a priori organization of the wireless ad hoc network in form of a rooted tree. For the purpose of producing such a tree with a low height, the distributed method described in [3] may be used. The result is an induced tree that includes all the nodes of the network and has minimum diameter. In comparison to the rest of the nodes, the root node (or master node) of the tree will have some additional tasks to fulfill. It runs, for example, a protocol to calculate the height of the tree. Moreover, the master node initiates the start of the FrogSim algorithm by means of a broadcast message. This message may additionally be used for communicating the height of the tree to the rest of the nodes as well. The induced tree is used during the execution of the FrogSim algorithm for communicating the node-color assignments to the master node and for calculating the state of convergence which will be used to stop the algorithm.

Algorithm 1 Program of each node $i \in V$

1: $\theta_i :=$ calculateNewThetaValue()
2: $c_i :=$ minimumColorNotUsed()
3: sendColoringMessage()
4: clearMessageQueue()

2.2 Main Algorithm

The main FrogSim algorithm works as follows. At each communication round (or iteration) each node executes the steps that are shown in Algorithm 1. The precise moment at which a node $i \in V$ starts executing this program depends on the value of variable $\theta_i \in [0, 1)$, which is stored internally. More precisely, assuming that the current communication round starts at time t, node i executes its program at time $t + \theta_i$. Apart from θ_i, each node i also maintains a color, denoted by $c_i \in \mathbb{N}^+$. Note that, for simplicity and without loss of generality, natural numbers greater than zero are used to uniquely identify colors. The execution of the program from Algorithm 1 includes the sending of exactly one message. In order to receive these messages from neighboring nodes, each node i maintains a message queue Q_i. In the following we provide a technical description of the functions of Algorithm 1.

When executing its program, a node i first adapts the value of θ_i in function calculateNewThetaValue(). This is done on the basis of the messages from the message queue Q_i. Only one message from each possible sender node is considered. In the case that Q_i contains two or more messages from the same sender node, the newest one prevails and the others are discarded. A message $m \in Q_i$ has the following format:

$$m =< \text{theta}_m, \text{color}_m > \ , \tag{1}$$

where $\text{theta}_m \in [0.1)$ contains the θ-value of the emitter and color_m is the color currently used by the emitter. Next, based on the messages in Q_i, function calculateNewThetaValue() calculates a new value for θ_i:

$$\theta_i := \theta_i - \alpha \sum_{m \in M_i} \frac{\mathbf{sin}(2\pi \cdot (\text{theta}_m - \theta_i))}{2\pi} \quad , \tag{2}$$

where $\alpha \in [0, 1]$ is a parameter used to control the convergence of the system. In general, the lower the value of α the smaller the change applied to θ_i.

Then, node i decides for a (possibly) new color in function minimumColorNotUsed(). Formally, this function computes the following value:

$$c_i := \mathsf{min}\{c \in \mathbb{N}^+ \mid \forall m \in Q_i \colon \text{color}_m \neq c\} \tag{3}$$

In words, node i chooses the color with the lowest identifier while discarding those colors that appear in messages $m \in Q_i$. Before finalizing its program,

node i must communicate its new color to its neighbors. This is done by means of function sendColoringMessage(). This function sends the following message m:

$$m =< \text{theta}_m := \theta_i, \text{color}_m := c_i > \tag{4}$$

To conclude the description of node i's program, the message queue Q_i is cleared by removing all messages (see function clearMessageQueue()).

2.3 Identifying the Best Coloring and Detecting Convergence

The way in which the algorithm identifies a new best coloring and detects convergence is based on the use of the induced tree structure which was generated at the start of the algorithm. In the following we provide a short description of the mechanism. For a complete technical description we refer the interested reader to [8, 9].

Henceforth, let h refer to the height—that is, the maximal distance between a leaf and the master node—of the induced tree. Note that h corresponds to the maximum number of communication rounds necessary for the master node to pass information to the rest of the nodes, and vice versa. In the following we assume that the master node knows about the size—in terms of the number of nodes—of the network. At each communication round, each node i is required to communicate the following information to its parent node in the induced tree: (1) a real number corresponding to the sum of the distances between the old theta values and the new ones concerning all nodes included in the subtree of which it acts as root, (2) the index of the largest color used by itself and all nodes included in the subtree of which it acts as root, and (3) an integer indicating the corresponding communication round number. In fact, these values do not need to be sent in extra messages. Instead they may be added to the coloring messages of Algorithm 1. Even though these messages will be received by all neighboring nodes, only the parent nodes in the induced tree will care about this information. Therefore, no additional messages are required by this mechanism.

Note that, in the first communication round, only the leaves of the tree will report the information described above to their parents. This is because the leaves are the only nodes without children. In the second communication round, the parents of the leaves will be able to report the aggregated data to their respective parents. Given the height h of the tree, it takes h communication rounds until all the information regarding a specific communication round has reached the master node. This means that the sensor nodes must store the differences between their old and new theta values, and the information about color use, during h communication rounds. Once the master node has received all the necessary information concerning a specific communication round, it is able to derive the following information. First, it knows the maximum index of any color used at the corresponding communication round. This information can be used to determine if a new best coloring has been found. Second, by dividing the sum of all theta-differences by the size of the network it obtains the average change of the theta-values in the corresponding communication round.

In case this average change is below a certain threshold value (we used 0.001), the master node broadcasts a stopping message to all nodes, which terminates the algorithm.

3 Experimental Evaluation

FrogSim was implemented in C++ without the use of any external libraries. Experiments were performed by means of discrete event simulation. As mentioned before, in this work we study the ability of the algorithm to generate colorings in which the number of nodes assigned to the most-used color is rather large. In other words, we study if large independent sets may be extracted from the colorings produced by the algorithm.[4] For this purpose we decided to test the algorithm on random geometric graphs, which are commonly used to model wireless ad hoc networks. For comparison we used the optimized version (from [10, 11]) of a very recent algorithm published in *Science* [2].

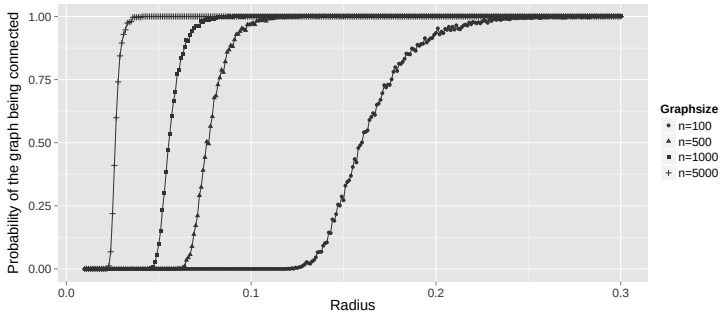
3.1 Generation of the Benchmark Set

Random geometric graphs are arguably the most popular model of wireless ad hoc networks. Therefore, we decided to study the algorithm in the context of this graph type. In order to generate a random geometric graph, one must first choose the number of nodes (n). These nodes are then assigned to random positions in the unit square. Finally, a fixed radius $0 < r < 1$ is used in order to determine the neighbors of each node. In particular, each pair of nodes that are within Euclidean distance of at most r are connected by an edge.

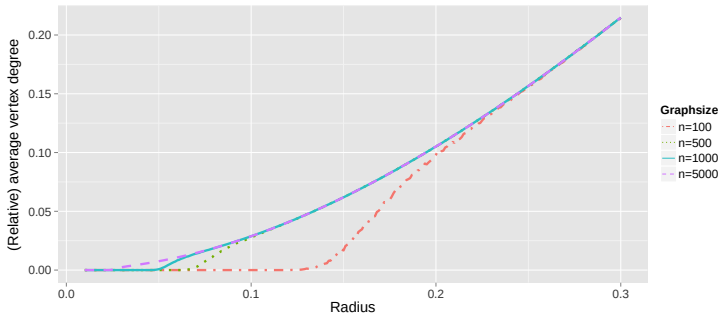
We considered random geometric graphs of sizes $n \in \{100, 500, 1000, 5000\}$. In order to find a reasonable range for the r-values for each n, the following experiments were performed. For each combination of $r \in \{0.01, 0.02, 0.03, \ldots, 0.3\}$ and n we generated 100 random geometric graphs and recorded the probability of these graphs to be connected. The results are graphically presented in Figure 1(a). Based on these results we determined the smallest r-values to be considered for the four graph sizes to be $r = 0.14$ (in case $n = 100$), $r = 0.067$ (in case $n = 500$), $r = 0.049$ (in case $n = 1000$) and $r = 0.024$ (in case $n = 5000$). With these values of r, the generated random geometric graphs have a probability of approx. 5% to be connected. Moreover, the resulting graphs are rather sparse.

In order to find suitable upper ranges for the r-values, we examined the *(relative) average degrees* of the generated graphs. Hereby, the term (relative) average degree refers to the average degree of a node expressed in terms of the fraction of all nodes to which the respective degree corresponds. For example, assume that $n = 100$ and that a node is, on average, connected to 10 neighbors. In this

[4] Note that, given a coloring solution, the nodes that are assigned to the same color form an independent set. Henceforth, given a coloring solution, we regard the size of the node set that is assigned to the most-used color as the MIS-value of the corresponding coloring.



(a) Probabilities for a graph being connected.



(b) (Relative) average degree of the nodes.

Fig. 1. Test results used for generating the test instances.

case the relative average degree is 0.1. In particular, we decided that the densest graphs to be considered in this study should have a (relative) average degree of 0.05. The graphic of Figure 1(b) shows that such graphs can be generated with $r = 0.169$ (in case $n = 100$) and $r = 0.134$ (in case $n \in \{500, 1000, 5000\}$).

For each value of n, numerical tests are performed for 20 values of r equidistantly distributed between the corresponding lower and the upper range.

3.2 Results

The numerical results are shown in Table 1 (graphs with $n \in \{100, 500\}$) and Table 2 (graphs with $n \in \{1000, 5000\}$). Each table row provides—for a certain combination of n and r, as indicated in the first two table columns—average results for 100 random geometric graphs. The column with heading Greedy provides the results for the most well-known (centralized) greedy algorithm for the MIS problem. This algorithm works as follows: at each iteration, first, it identifies the node with minimal degree and adds it to the maximal independent set under construction. Afterwards, this node—together with all its neighboring nodes—is removed from the input graph. This procedure stops once the input graph is empty. The results of this algorithm are simply given in order to in-

Table 1. Results for random geometric graphs with 100 and 500 nodes.

#nodes (n)	**radius** (r)	GREEDY	FRUITFLY		FROGSIM		
			avg.	**rounds**	**avg.**	**rounds**	**convergence**
100	0.14	30.22	27.97	27.86	29.16	244.24	711.35
	0.1415	29.90	27.38	28.92	28.42	203.45	687.60
	0.143	29.47	27.19	27.10	27.96	205.23	691.00
	0.1445	28.99	26.76	29.04	27.58	161.85	678.48
	0.146	28.63	26.17	31.96	26.96	168.29	674.39
	0.1475	28.22	25.86	28.96	26.86	212.75	708.41
	0.149	28.06	25.74	34.30	26.46	179.64	708.32
	0.1505	27.52	25.44	30.48	26.42	221.74	707.78
	0.152	27.27	24.94	31.86	25.89	159.11	700.75
	0.1535	26.89	24.90	32.34	25.81	202.76	693.56
	0.155	26.69	24.59	32.86	25.51	180.87	708.80
	0.1565	26.41	24.29	33.02	25.19	155.20	699.52
	0.158	25.93	23.68	36.54	24.58	169.57	725.95
	0.1595	25.51	23.72	36.52	24.41	216.11	742.00
	0.161	25.24	23.20	36.68	24.09	159.63	706.00
	0.1625	24.99	22.85	35.58	23.76	167.35	699.47
	0.164	24.79	22.56	36.14	23.67	172.80	713.10
	0.1655	24.63	22.70	37.96	23.57	207.55	736.86
	0.167	24.28	22.21	42.18	23.22	187.38	739.65
	0.169	23.80	21.93	38.72	22.77	166.64	708.97
500	0.067	130.20	120.19	52.00	122.52	341.22	726.23
	0.0705	121.42	111.54	62.24	114.19	375.81	740.82
	0.074	113.10	103.57	64.42	106.78	351.93	739.05
	0.0775	105.80	97.99	78.44	100.53	345.00	753.65
	0.081	98.79	91.56	90.98	93.94	312.48	764.53
	0.0845	92.72	86.28	98.70	88.47	305.55	776.57
	0.088	87.38	81.56	126.48	83.22	328.97	778.19
	0.0915	82.40	76.73	151.40	78.46	267.00	771.31
	0.095	77.46	72.31	191.62	73.68	265.35	767.13
	0.0985	73.06	68.73	219.38	69.96	280.98	788.11
	0.102	69.51	65.37	309.80	66.49	264.84	762.38
	0.1055	65.70	62.38	374.94	62.90	266.77	771.17
	0.109	62.41	59.28	430.32	59.81	236.35	756.69
	0.1125	59.30	56.72	534.84	56.98	259.36	746.62
	0.116	56.47	54.28	626.82	54.39	227.10	770.09
	0.1195	53.69	52.18	777.42	51.91	205.91	757.79
	0.123	51.31	49.65	*1043.28*	49.74	222.74	784.69
	0.1265	49.04	47.43	*1294.66*	47.37	193.61	760.04
	0.13	46.8	46.07	*1663.82*	45.64	206.74	752.46
	0.134	44.71	44.32	*2148.60*	43.49	199.54	724.70

dicate the quality of our algorithm in comparison to a centralized technique. The next two columns of the result tables provide the results of the most recent distributed algorithm for the MIS problem [2], which was inspired by the way in which neural precursors are selected during the development of the nervous system of the fruit fly *Drosophila*. We implemented an optimized version of this algorithm—published in [10, 11]—which is henceforth referred to as FRUITFLY. The results of FRUITFLY are given in two columns. The first one, with heading **avg.**, provides the average result over 100 random geometric graphs. The second column, with heading **rounds**, indicates the average number of communication rounds that were performed in order for the algorithm to finish. Finally, the last three table columns contain the results of FROGSIM. As in the case of FRUIT-

Table 2. Results for random geometric graphs with 1000 and 5000 nodes.

#nodes (n)	**radius** (r)	GREEDY	FRUITFLY		FROGSIM		
			avg.	**rounds**	**avg.**	**rounds**	**convergence**
1000	0.049	244.88	225.39	66.66	229.76	416.14	734.08
	0.0534	215.26	199.07	80.78	203.48	442.16	750.92
	0.0578	190.96	176.50	115.30	180.26	414.50	758.72
	0.0622	170.02	158.01	160.02	161.35	366.59	768.56
	0.0666	152.35	142.98	236.02	144.82	325.74	775.16
	0.071	137.45	130.18	315.16	130.68	322.37	767.18
	0.0754	124.53	118.74	480.64	118.82	272.97	770.88
	0.0798	113.26	109.25	721.16	108.32	274.80	767.69
	0.0842	103.82	100.91	*1114.90*	99.55	248.03	754.76
	0.0886	95.35	93.68	*1865.20*	91.75	268.64	743.73
	0.093	87.82	87.19	*2562.90*	84.93	279.20	755.75
	0.0974	81.13	81.63	*4631.68*	78.48	252.60	747.29
	0.1018	75.42	76.72	*7014.74*	73.16	212.85	750.31
	0.1062	70.55	71.77	*11754.56*	68.23	216.22	712.49
	0.1106	65.61	67.91	*22063.76*	64.14	205.49	740.14
	0.115	61.70	63.94	*34864.16*	60.33	245.18	715.50
	0.1194	57.84	60.60	*53523.54*	56.55	215.69	684.79
	0.1238	54.67	57.24	*96351.26*	53.52	178.10	699.82
	0.1282	51.53	54.60	*165323.04*	50.16	165.77	700.42
	0.134	47.83	50.84	*321192.92*	46.95	160.96	678.45
5000	0.024	1040.74	962.55	129.46	976.72	561.54	761.83
	0.0297	736.45	694.10	370.78	694.91	498.26	776.10
	0.0354	543.55	528.05	*1211.48*	515.79	431.58	771.42
	0.0411	419.17	420.54	*4542.54*	401.35	380.55	766.95
	0.0468	331.90	345.53	*23172.80*	323.11	409.65	752.35
	0.0525	269.66	289.20	*131172.26*	265.49	415.70	740.25
	0.0582	224.68	245.77	*749947.06*	222.63	353.63	722.16
	0.0639	190.44	75.27	*1000000.00*	189.05	321.00	718.15
	0.0696	163.88	0.00	*1000000.00*	163.21	344.87	737.77
	0.0753	141.95	0.00	*1000000.00*	142.00	311.10	731.58
	0.081	124.59	0.00	*1000000.00*	125.26	331.54	729.67
	0.0867	110.54	0.00	*1000000.00*	111.30	300.39	703.93
	0.0924	98.90	0.00	*1000000.00*	99.68	278.69	690.82
	0.0981	89.16	0.00	*1000000.00*	89.76	262.51	688.26
	0.1038	80.80	0.00	*1000000.00*	81.28	218.35	676.89
	0.1095	73.94	0.00	*1000000.00*	74.30	267.05	689.92
	0.1152	67.84	0.00	*1000000.00*	67.75	191.87	660.59
	0.1209	62.20	0.00	*1000000.00*	62.50	182.83	640.03
	0.1266	57.66	0.00	*1000000.00*	57.81	201.70	606.51
	0.134	52.34	0.00	*1000000.00*	52.33	140.24	590.34

FLY, the first column provides the average result for the respective 100 random geometric graphs. The second column, with heading **rounds**, provides the average number of communication rounds that were performed in order to achieve the results reported in the column with label **avg.**. The last column, with heading **convergence**, indicates the average number of communication rounds that were performed in order for the algorithm to converge. Finally, note that the best result of each table row is marked with a grey background.

Based on the results that are displayed in Tables 1 and 2, the following observations can be made. First, concerning graphs of sizes $n \in \{100, 500\}$, FROGSIM

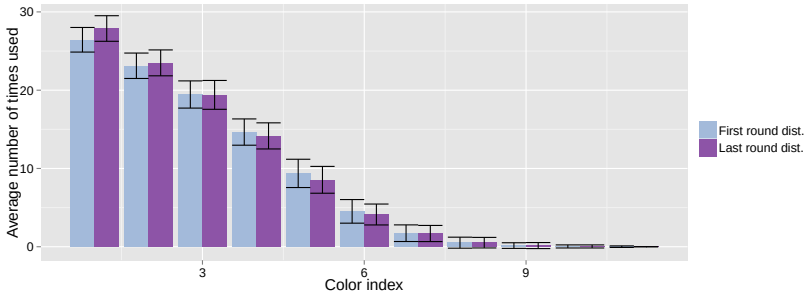
outperforms FruitFly consistently, with the exception of rather dense graphs of size 500—that is, graphs generated with a radius r tending towards the upper range—where FruitFly seems to provide slightly better results. However, when consulting the number of communication rounds needed by FruitFly it becomes clear that with growing graph size and density, the communication round requirements grow significantly. In fact, we underlined all cases in which FruitFly needs more than 1000 communication rounds for providing a result. Moreover, all runs of FruitFly were performed with a maximum of one million communication rounds. In contrast to FruitFly, the communication round requirements of FrogSim do not seem to depend on the graph size. In any case, the communication round requirements of FrogSim even seem to decrease with growing graph density. This is certainly a desirably property of a distributed algorithm.

The results for graph sizes $n \in \{1000, 5000\}$ amplify the observations outlined above. In fact, FrogSim still seems to work better than FruitFly for what concerns the sparsest graphs. However, starting from $r = 0.0798$ (in the case of $n = 1000$) and $r = 0.0354$ (in the case of $n = 5000$) FruitFly starts to produce better results than FrogSim. However, the number of communication rounds necessary for beating FrogSim quickly becomes unpractical. For example, when $n = 5000$, FruitFly is not able to provide results within one million communication rounds for the whole range of $r \in [0.0639, 0.134]$.

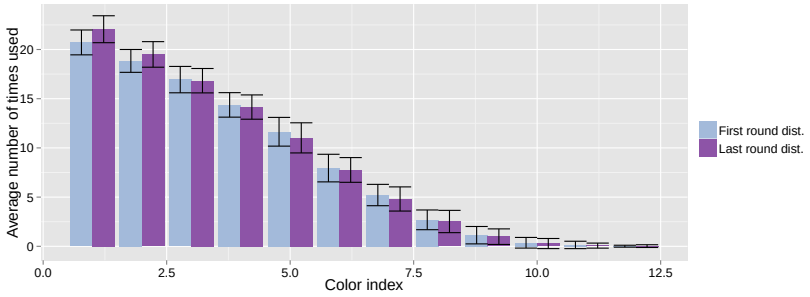
Finally, we aim at studying the thrive of FrogSim towards colorings in which the *most-used color* corresponds to rather large independent sets. For this purpose we examined the results of FrogSim for what concerns the lowest and the highest setting of the radius r for all four graph sizes. In particular, for each of these cases we display the color distribution (averaged over 100 random geometric graphs) after the first communication round in contrast to the color distribution after convergence. This information is shown in Figures 2 and 3. On the x-axis of these graphics we can find the indices of the used colors. The bars (including the standard deviation) indicate for each color index the number of nodes that have assigned the respective color. For example, the graphic in Figure 2(c) shows that—in the case $n = 500$, $r = 0.067$—the color with the lowest index is used by around 112 nodes (on average) after the first communication round. In contrast, the same color is used by around 125 nodes (on average) after the last communication round. This clearly indicates the thrive of the algorithm towards the creation of colorings in which the most-used color corresponds to large independent sets. Moreover, the eight graphics indicate that this is a general trend, independent of graph size and density.

4 Conclusions and Future Work

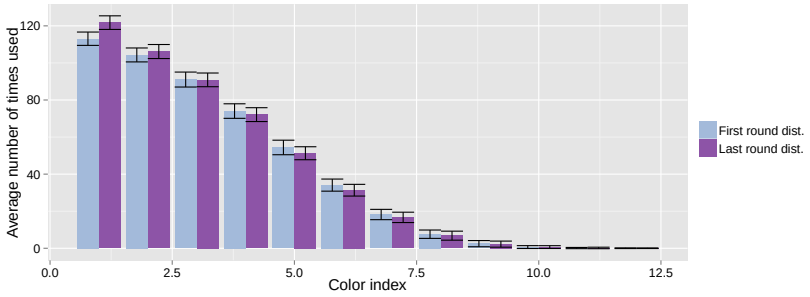
In this work we studied a bio-inspired, distributed algorithm—initially introduced for graph coloring—for its ability to generate graph coloring solutions in which the independent set of nodes assigned to the most-used color is large. The



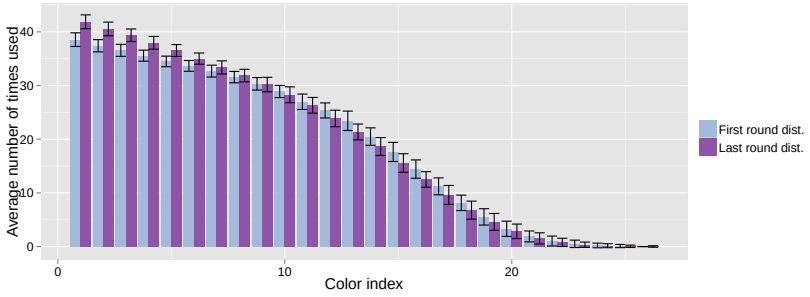
(a) $n = 100$, radius 0.14.



(b) $n = 100$, radius 0.169.

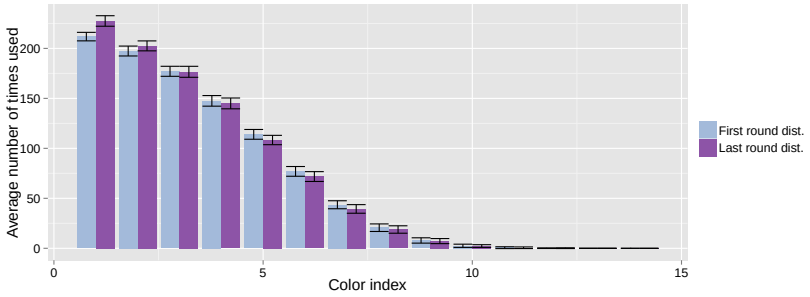


(c) $n = 500$, radius 0.067.

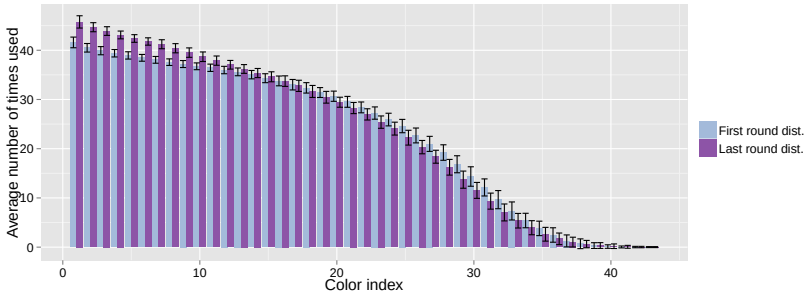


(d) $n = 500$, radius 0.134.

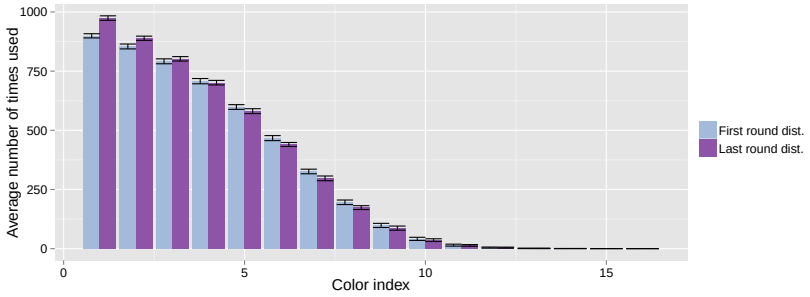
Fig. 2. The graphics show the distribution of the use of the different colors both at the start of the algorithm (see *First round dist.*) and after convergence (see *Last round dist.*) for graphs of sizes 100 and 500, and different values of r.



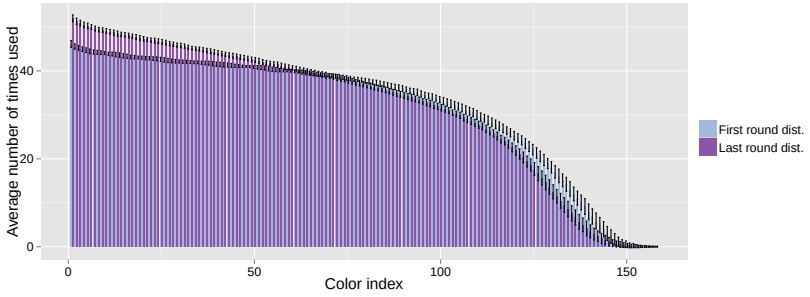
(a) $n = 1000$, radius 0.049.



(b) $n = 1000$, radius 0.134.



(c) $n = 5000$, radius 0.024.



(d) $n = 5000$, radius 0.134.

Fig. 3. The graphics show the distribution of the use of the different colors both at the start of the algorithm (see *First round dist.*) and after convergence (see *Last round dist.*) for graphs of sizes 1000 and 5000, and different values of r.

results, in terms of the size of the independent set that is generated, were compared to the most recent algorithm published in the related literature for the maximal independent set problem. They show that the algorithm performs especially well in the context of sparse graphs. An important additional advantage is to be found in the low number of required communication rounds. The algorithm always converges within a reasonable number of communication rounds, independent of graph size and density.

Future work will focus on the study of the performance of the algorithm on different types of graphs. Moreover, we will study ways for improving the algorithms' performance for dense graphs.

Acknowledgments. This work was supported by projects TIN2012-37930, TIN2010-14931 and TIN2007-66523 of the Spanish Government, and project 2009-SGR1137 of the Generalitat de Catalunya. In addition, support is acknowledged from IKERBASQUE (Basque Foundation for Science) and the Basque Saiotek and Research Groups 2013-2018 (IT-609-13) programs.

References

1. Online scientific news site Science*Daily*: Frog calls inspire a new algorithm for wireless networks (July 2012), http://www.sciencedaily.com/releases/2012/07/120717100123.htm
2. Afek, Y., Alon, N., Barad, O., Hornstein, E., Barkai, N., Bar-Joseph, Z.: A biological solution to a fundamental distributed computing problem. Science 331, 183–185 (2011)
3. Bui, M., Butelle, F., Lavault, C.: A distributed algorithm for constructing a minimum diameter spanning tree. Journal of Parallel and Distributed Computing 64(5), 571–577 (2004)
4. Erciyes, K.: Distributed Graph Algorithms for Computer Networks. Springer, London, UK (2013)
5. Erciyes, K., Dagdeviren, O., Cokuslu, D., Yilmaz, O., Gumus, H.: Modeling and Simulation of Mobile Ad hoc Networks, pp. 134–168. CRC Press (2010)
6. Garey, M.R., Johnson, D.S.: Computers and intractability; a guide to the theory of *NP*-completeness. W. H. Freeman (1979)
7. Halldórsson, M.M., Radhakrishnan, J.: Greedy is Good: Appriximating Independent Sets in Sparse and Bounded-Degree Graphs. Algorithmica 18, 145–163 (1997)
8. Hernández, H., Blum, C.: Distributed graph coloring: An approach based on the calling behavior of japanese tree frogs. Swarm Intelligence 6(2), 117–150 (2012)
9. Hernández, H., Blum, C.: FrogSim: distributed graph coloring in wireless ad hoc networks — an algorithm inspired by the calling behavior of Japanese tree frogs. Telecommunication Systems (2013)
10. Scott, A., Jeavons, P., Xu, L.: Feedback from nature: an optimal distributed algorithm for maximal independent set selection. Tech. rep., ArXiv repository, http://arxiv.org/abs/1211.0235 (2012)
11. Scott, A., Jeavons, P., Xu, L.: Feedback from nature: an optimal distributed algorithm for maximal independent set selection. In: Fatourou, P., Taubenfeld, G. (eds.) Proceedings of PODC 2013 – ACM Symposium on Principles of Distributed Computing. pp. 147–156. ACM Press (2013)